МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ

по дисциплине **«Распределенные вычисления»**
для студентов направления подготовки
02.03.01«Математика и компьютерные науки»
направленность (профиль)
«Анализ данных и искусственный интеллект»

Введение

Лабораторные работы - это целенаправленная форма организации педагогического процесса, направленная на углубление научно-теоретических знаний и овладение определенными методами работы, в процессе которых вырабатываются умения и навыки выполнения тех или иных учебных действий в данной сфере науки.

Лабораторные работы предназначены для углубленного изучения учебных дисциплин и играют важную роль в выработке у студентов умений и навыков применения полученных знаний для решения практических задач совместно с педагогом. Кроме того, они развивают научное мышление и речь, позволяют проверить знания студентов и выступают как средства оперативной обратной связи.

Цель лабораторных работ - углублять, расширять, детализировать знания, полученные на лекции, в обобщенной форме и содействовать выработке навыков профессиональной деятельности.

Данные методические указания к лабораторным работам разработаны в соответствии с рабочей программой по дисциплине «Распределенные вычисления» для студентов направления подготовки 02.03.01 «Математика и компьютерные науки».

Лабораторная работа 1

Тема: «**Система контроля версий Git**».

Цель: Ознакомиться с системой контроля версий Git. Изучить основные возможности Git и принципы его работы. Приобрести практические навыки работы с Git для ведения проектов, включая создание репозитория, добавление и коммит изменений, ветвление, слияние и разрешение конфликтов. Ознакомиться с современными клиентами Git и их использованием. Рассмотреть применение Git для управления проектами в распределенных командах.

Краткая теоретическая справка

Система контроля версий (Version Control System, VCS) – это инструмент, предназначенный для отслеживания изменений в файлах проекта с течением времени. Он позволяет возвращаться к предыдущим версиям, сравнивать изменения и координировать работу нескольких разработчиков над одним проектом. Функция input() принимает **текст**, который пользователь вводит с клавиатуры.

Git – это распределенная система контроля версий, разработанная Линусом Торвальдсом. Основные характеристики Git:

- Распределенность: Каждый разработчик имеет полную копию репозитория, что позволяет работать автономно и повышает отказоустойчивость.
- Скорость: Git спроектирован для быстрой работы с большими проектами.
- Ветвление и слияние: Ветви – это мощный механизм для разработки новых функций или исправления ошибок изолированно от основной кодовой базы. Слияние позволяет объединять изменения из разных ветвей.
- Поддержка нелинейной разработки: Git позволяет разрабатывать параллельно несколько версий проекта.
- Безопасность: Git использует криптографические хэши для обеспечения целостности данных.

Основные команды Git:

- gitinit: Инициализация нового репозитория Git.
- gitclone: Клонирование существующего репозитория.
- gitadd: Добавление файлов в область подготовленных изменений (stagingarea).
- gitcommit: Фиксация изменений в репозитории.
- gitpush: Отправка изменений на удаленный репозиторий.

- gitpull: Получение изменений с удаленного репозитория.
- gitbranch: Создание, просмотр и удаление веток.
- gitcheckout: Переключение между ветками.
- gitmerge: Слияние веток.
- gitlog: Просмотр истории коммитов.
- gitstatus: Просмотр текущего состояния репозитория.
- gitdiff: Просмотр изменений между коммитами, рабочей областью и областью подготовленных изменений.

Задание 1. Основы работы с Git (командная строка)

1. Создание локального репозитория:
    a. Создайте новую директорию для проекта.
    b. Перейдите в эту директорию через командную строку.
    c. Инициализируйте новый репозиторий Git с помощью команды gitinit.
2. Добавление и фиксация изменений:
    a. Создайте текстовый файл README.md с кратким описанием проекта.
    b. Добавьте файл README.md в область подготовленных изменений с помощью команды gitadd README.md.
    c. Зафиксируйте изменения с помощью команды gitcommit -m "Initialcommit: Added README.md".
3. Просмотр истории коммитов:
    a. Просмотрите историю коммитов с помощью команды gitlog.
4. Изменение файла и создание нового коммита:
    a. Отредактируйте файл README.md, добавив дополнительную информацию о проекте.
    b. Добавьте измененный файл в область подготовленных изменений.
    c. Зафиксируйте изменения с помощью команды gitcommit -m "Updated README.md withmoredetails".

5. Просмотр статуса репозитория:
    a. Проверьте текущий статус репозитория с помощью команды gitstatus.
6. Сравнение изменений:
    a. Сравните текущую версию файла README.md с предыдущей версией с помощью команды gitdiff.

Задание 2. Ветвление и слияние

1. Создание новой ветки:
    a. Создайте новую ветку с именем feature/new-feature с помощью команды git branch feature/new-feature.
    b. Переключитесь на новую ветку с помощью команды gitcheckoutfeature/new-feature.
2. Внесение изменений в новой ветке:
    a. Создайте новый файл new_feature.txt с каким-либо содержанием.
    b. Добавьте и зафиксируйте изменения в новой ветке.
3. Переключение на главную ветку:
    a. Переключитесь обратно на главную ветку (main или master).
4. Слияние ветки:
    a. Слейте ветку feature/new-feature с главной веткой с помощью команды git merge feature/new-feature.
5. Разрешение конфликтов (если возникли):
    a. Если при слиянии возникли конфликты, разрешите их вручную, отредактировав файлы и указав, какие изменения следует сохранить.
    b. После разрешения конфликтов добавьте изменения в область подготовленных изменений и зафиксируйте их.
6. Удаление ветки:
    a. Удалите ветку feature/new-feature после успешного слияния с помощью команды gitbranch -d feature/new-feature.

Задание 3. Работа с удаленным репозиторием (GitHub/ GitLab/ Bitbucket)

1. Создание удаленного репозитория:
    a. Создайте новый репозиторий на GitHub, GitLab или Bitbucket.
2. Связывание локального репозитория с удаленным:
    a. Добавьте удаленный репозиторий к локальному репозиторию с помощью команды gitremoteaddorigin<URL вашего удаленного репозитория>.
3. Отправка изменений на удаленный репозиторий:
    a. Отправьте изменения из локального репозитория на удаленный репозиторий с помощью команды gitpush -u originmain (или master).
4. Клонирование удаленного репозитория:
    a. Создайте новую директорию.
    b. Клонируйте удаленный репозиторий в эту директорию с помощью команды gitclone<URL вашего удаленного репозитория>.

Задание 4. Использование графического клиента Git (по выбору)

1. Установите и настройте один из графических клиентов Git (GitHubDesktop, GitKraken, SourceTree, SmartGit).
2. Выполните все предыдущие задания, используя графический клиент.
3. Сравните удобство работы с Git через командную строку и через графический клиент.

Контрольные вопросы:

1. Что такое система контроля версий и зачем она нужна?
2. Что такое Git и каковы его основные преимущества?
3. Что такое коммит, ветка и слияние в Git?
4. Как разрешать конфликты при слиянии веток?
5. Как создать и использовать удаленный репозиторий?
6. Какие графические клиенты Git вы знаете и какие у них преимущества?
7. Как Git помогает управлять проектами в распределенных командах?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Скриншоты, иллюстрирующие выполнение заданий (особенно разрешение конфликтов). Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 2

Тема: «**Основы и синтаксис языка Java**»

Цель: Ознакомиться с базовым синтаксисом и основными понятиями языка программирования Java. Приобрести практические навыки написания простых Java-программ. Получить представление о фреймворках, используемых для построения распределенных систем на Java. Выполнить простую программу, демонстрирующую базовый синтаксис Java.

Теоретическая справка (переписать в тетрадь)

Java – это объектно-ориентированный язык программирования, разработанный компанией SunMicrosystems (позже приобретенной Oracle). Основные характеристики Java:

1. Объектно-ориентированность (ООП): Java поддерживает принципы ООП, такие как инкапсуляция, наследование и полиморфизм.
2. Платформонезависимость: "WriteOnce, RunAnywhere" (WORA) – код Java компилируется в байт-код, который выполняется на виртуальной машине Java (JVM), обеспечивая переносимость.
3. Безопасность: Java предоставляет механизмы безопасности, предотвращающие выполнение вредоносного кода.
4. Многопоточность: Java поддерживает многопоточное программирование, позволяя выполнять несколько задач параллельно.
5. Автоматическое управление памятью: Сборщик мусора (garbagecollector) автоматически освобождает память, занимаемую неиспользуемыми объектами.

Основные понятия Java:

1. Классы (Classes): Шаблоны для создания объектов.
2. Объекты (Objects): Экземпляры классов.
3. Методы (Methods): Функции, определяющие поведение объектов.
4. Переменные (Variables): Хранят данные.
5. Типы данных (DataTypes): Определяют тип данных, которые может хранить переменная (например, int, double, String, boolean).

6. Операторы (Operators): Символы, выполняющие операции над данными (например, +, -, *, /, ==, !=).
7. Управляющие конструкции (ControlFlow): Определяют порядок выполнения кода (if-else, for, while).
8. Пакеты (Packages): Используются для организации классов в логические группы.

Задание 1. Написание простой Java-программы:

Создайте класс HelloWorld с методом main. В методе main выведите на консоль строку "Hello, World!". Скомпилируйте и запустите программу.

Задание 2. Работа с переменными и типами данных:

Объявите переменные различных типов данных (int, double, boolean, String). Присвойте переменным значения. Выведите значения переменных на консоль. Выполните арифметические операции с переменными типа int и double.

Задание 3. Управляющие конструкции:

Напишите программу, которая запрашивает у пользователя число и определяет, является ли оно четным или нечетным, используя конструкцию if-else. Напишите программу, которая выводит на консоль таблицу умножения для заданного числа, используя цикл for. Напишите программу, которая запрашивает у пользователя числа до тех пор, пока он не введет 0, используя цикл while.

Задание 4. Работа с массивами:

Создайте массив целых чисел. Заполните массив случайными числами. Выведите элементы массива на консоль. Найдите минимальный и максимальный элементы массива. Отсортируйте массив по возрастанию.

Задание 5. Создание класса:

Создайте класс Rectangle с полями width и height. Создайте конструктор, который принимает значения width и height. Создайте методы getArea() и getPerimeter(), которые возвращают площадь и периметр прямоугольника соответственно. Создайте объект класса Rectangle и выведите его площадь и периметр на консоль.

Задание 6. Обзор фреймворков для распределенных систем:

Изучите документацию ApacheKafka, SpringCloud. Сравните ApacheKafka и SpringCloud по следующим критериям:

- Назначение
- Архитектура
- Сложность
- Примеры использования

Контрольные вопросы:

1. Что такое JavaVirtual Machine (JVM) и зачем она нужна?
2. Что такое классы и объекты в Java?
3. Какие типы данных существуют в Java?
4. Какие управляющие конструкции существуют в Java?
5. Что такое пакеты в Java и зачем они нужны?
6. Что такое распределенная система?
7. Какие фреймворки используются для построения распределенных систем на Java?
8. Опишите основные компоненты Apache Kafka и Spring Cloud.
9. В каких случаях целесообразно использовать Apache Kafka, а в каких SpringCloud?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Код программ, написанных в первой части работы. Описание выполненных заданий. Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 3

Тема: **«Основные принципы параллельных вычислений на примере языка Java».**

Цель: Изучить основные принципы параллельных вычислений. Освоить методы распараллеливания алгоритмов. Приобрести практические навыки использования многопоточности в Java с помощью встроенных классов. Ознакомиться с пакетом java.util.concurrent и его возможностями. Научиться оценивать эффективность параллельных вычислений.

Краткая теоретическая справка

Параллельные вычисления – это метод выполнения вычислений, при котором задача разбивается на несколько подзадач, которые выполняются одновременно на нескольких вычислительных устройствах (например, ядрах процессора).

Многопоточность – это возможность выполнения нескольких потоков (threads) в рамках одного процесса.

При работе с многопоточностью необходимо обеспечить синхронизацию потоков, чтобы избежать гонок данных и других проблем. Пакет java.util.concurrent предоставляет набор классов и интерфейсов для упрощения разработки многопоточных приложений.

Задание 1. Создание нескольких потоков:

Создайте класс MyThread , который наследуется от класса Thread .Переопределите метод run() в классе MyThread .В методе run() выведите на консоль сообщение "Thread [номер потока] isrunning".Создайте несколько объектов класса MyThread и запустите их.

Задание 2. Использование Runnable:

Создайте класс MyRunnable, который реализует интерфейс Runnable. Реализуйте метод run() в классе MyRunnable.В методе run() выведите на консоль сообщение "Runnable [номер потока] isrunning".Создайте несколько объектов класса Thread, передав в конструктор объекты класса MyRunnable, и запустите их.

Задание 3. Синхронизация потоков:

Создайте метод increment() , который увеличивает значение поля count на 1.Создайте несколько потоков, которые вызывают метод increment() класса Counter. Используйте ключевое слово synchronized для защиты метода increment () от одновременного доступа нескольких потоков. Проверьте, что значение поля count после выполнения всех потоков равно ожидаемому.

Задание 4. Использование ExecutorService:

Создайте пул потоков с помощью класса Executors. Создайте несколько задач, реализующих интерфейс Runnable или Callable. Отправьте задачи на выполнение в пул потоков с помощью метода submit(). Дождитесь завершения выполнения всех задач с помощью метода shutdown() и awaitTermination().

Задание 5. Использование Future:

Создайте задачу, реализующую интерфейс Callable. Отправьте задачу на выполнение в пул потоков с помощью метода submit(). Получите объект Future, представляющий результат выполнения задачи. Дождитесь завершения выполнения задачи и получите результат с помощью метода get().

Задание 6. Использование ConcurrentHashMap:

Создайте объект класса ConcurrentHashMap. Создайте несколько потоков, которые добавляют и удаляют элементы из ConcurrentHashMap. Проверьте, что ConcurrentHashMap работает корректно в многопоточной среде.

Задание 7. Распараллеливание вычисления суммы элементов массива:

Реализуйте последовательный алгоритм вычисления суммы элементов массива. Реализуйте параллельный алгоритм вычисления суммы элементов массива, используя разделение данных (DataParallelism). Сравните время выполнения последовательного и параллельного алгоритмов для массивов разного размера.

Контрольные вопросы:

1. Что такое параллельные вычисления и зачем они нужны?
2. Какие методы распараллеливания алгоритмов вы знаете?
3. Что такое многопоточность в Java?
4. Как создать и запустить поток в Java?

5. Как обеспечить синхронизацию потоков в Java?
6. Какие классы и интерфейсы предоставляет пакет java.util.concurrent?
7. Что такое пул потоков и зачем он нужен?
8. Что такое Future и как его использовать?
9. Какие потокобезопасные коллекции существуют в Java?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Код программ, написанных. Описание выполненных заданий. Результаты сравнения времени выполнения последовательного и параллельного алгоритмов.Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 4

Тема: **Основы работы с сетью на примере языка Java**.

Цель: Изучить основные принципы работы с сетью на языке Java. Освоить встроенные методы Java для сетевого взаимодействия. Ознакомиться с различными библиотеками для работы с сетью в Java. Приобрести практические навыки создания простых сетевых приложений (клиент-сервер).

Краткая теоретическая справка.

Сетевое взаимодействие – это процесс обмена данными между двумя или более устройствами, подключенными к сети. В основе сетевого взаимодействия лежат протоколы, которые определяют правила обмена данными.

Пример простого TCP-сервера:

```
import java.net.*;
import java.io.*;

public class SimpleTCPServer {
   public static void main(String[] args) throws IOException {
      int port = 12345; // Порт для прослушивания
ServerSocketserverSocket = new ServerSocket(port);
System.out.println("Server started on port " + port);

      while (true) {
         Socket clientSocket = serverSocket.accept(); // Ожидание подключения клиента
System.out.println("Client connected: " + clientSocket.getInetAddress().getHostAddress());

            try (
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            ) {
               String inputLine;
               while ((inputLine = in.readLine()) != null) {
System.out.println("Received from client: " + inputLine);
out.println("Server received: " + inputLine); // Отправка ответа клиенту
```

```
                }
            } catch (IOException e) {
        System.err.println("Exception handling client: " + e.getMessage());
            } finally {
        clientSocket.close();
            }
        }
    }
}
```

Пример простого TCP-клиента:

```
import java.net.*;
import java.io.*;

public class SimpleTCPClient {
    public static void main(String[] args) throws IOException {
        String hostName = "localhost"; // Адресс сервера
        int portNumber = 12345; // Порт сервера

        try (
            Socket socket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        BufferedReaderstdIn = new BufferedReader(new InputStreamReader(System.in));
        ) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
        out.println(userInput); // Отправка данных на сервер
        System.out.println("Serverresponse: " + in.readLine()); // Получение ответа от сервера
            }
        } catch (UnknownHostException e) {
        System.err.println("Don't know about host " + hostName);
        System.exit(1);
        } catch (IOException e) {
```

```
            System.err.println("Couldn't get I/O for the connection to " + hostName);
            System.exit(1);
        }
    }
}
```

Задание 1. Простой TCP-сервер:

Напишите TCP-сервер, который принимает соединения от клиентов, получает от них сообщения и отправляет им ответное сообщение, содержащее полученное сообщение в верхнем регистре.

Напишите TCP-клиент, который подключается к серверу, отправляет ему сообщения, введенные пользователем с клавиатуры, и выводит на экран ответные сообщения, полученные от сервера.

Задание 2. Простой UDP-сервер:

Напишите UDP-сервер, который принимает UDP-пакеты от клиентов и выводит их содержимое на экран.

Напишите UDP-клиент, который отправляет UDP-пакеты на сервер, содержащие сообщения, введенные пользователем с клавиатуры.

Задание 3. ApacheHttpClient:

Напишите программу, которая отправляет HTTPGET-запрос на заданный URL и выводит содержимое ответа на экран, используя библиотеку ApacheHttpClient.

Задание 4. Использование Netty:

Напишите простой HTTP-сервер, использующий Netty.

Контрольные вопросы:

1. Что такое TCP и UDP? В чем их различия?
2. Что такое сокет?
3. Как создать TCP-сервер и TCP-клиент в Java?
4. Как создать UDP-сервер и UDP-клиент в Java?
5. Какие библиотеки для работы с сетью существуют в Java?
6. Что такое HTTP?
7. Как отправить HTTP GET-запрос с помощью ApacheHttpClient?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Код программ, написанных. Описание выполненных заданий. Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 5

Тема: **Основы сетевых топологий.**

Цель: Изучить основные сетевые топологии. Рассмотреть преимущества и недостатки различных топологий. Ознакомиться с организацией и возможностями применения топологий для распределения вычислений. Приобрести навыки анализа и выбора подходящей топологии для конкретной задачи.

Краткая теоретическая справка.

Сетевая топология – это физическое или логическое расположение узлов (компьютеров, серверов, сетевых устройств) и соединений (каналов связи) в сети. Выбор топологии влияет на производительность, надежность, масштабируемость и стоимость сети.

Основные сетевые топологии: Шина (Bus), Звезда (Star), Кольцо (Ring), Дерево (Tree), Ячеистая (Mesh), Гибридная (Hybrid).

Задание 1. Анализ и сравнение топологий:

Заполните таблицу, сравнивая различные топологии по следующим критериям:

Топология	Простота реализаций	Стоимость	Надёжность	Пропускная способность	Масштабируемость	Сложность диагностики
Шина						
Звезда						
Кольцо						
Дерево						
Ячеистая						

Задание 2.

Для каждой топологии приведите пример ситуации, когда ее использование наиболее целесообразно

Задание 3. Проектирование сети:

Представьте, что вам необходимо спроектировать сеть для офиса, в котором 20 компьютеров. Обоснуйте выбор топологии и укажите необходимое оборудование. Представьте, что вам необходимо спроектировать сеть для кластера, предназначенного для выполнения сложных научных расчетов. Обоснуйте выбор топологии и укажите необходимое оборудование. Представьте, что вам необходимо спроектировать сеть для системы видеонаблюдения, состоящей из 50 камер. Обоснуйте выбор топологии и укажите необходимое оборудование.

Задание 4. Моделирование сети (использование программного обеспечения)

Используйте программное обеспечение для моделирования сетей (например, CiscoPacketTracer, GNS3, EVE-NG) для создания модели сети с топологией "Звезда". Настройте IP-адреса для устройств в сети. Проверьте связь между устройствами с помощью команды ping. Смоделируйте отказ одного из узлов и убедитесь, что остальные узлы продолжают работать. Создайте модель сети с топологией "Ячеистая". Настройте маршрутизацию между узлами. Смоделируйте отказ одного из каналов связи и убедитесь, что данные продолжают передаваться по другим путям.

Контрольные вопросы:

1. Что такое сетевая топология?
2. Какие основные типы сетевых топологий вы знаете?
3. В чем преимущества и недостатки каждой топологии?
4. Как выбор топологии влияет на надежность, пропускную способность и масштабируемость сети?
5. Как различные топологии подходят для распределения вычислений?
6. Какие программные средства можно использовать для моделирования сетей?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Обоснование выбора топологии для каждой из предложенных задач проектирования. Скриншоты моделей сетей, созданных в программном обеспечении. Описание процесса настройки и проверки связи между устройствами. Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 6

Тема: **Основы современных фреймворков.**

Цель: Ознакомиться с концепциями и применением современных фреймворков для обработки больших данных, контейнеризации и работы с GPU. Приобрести практические навыки работы с Hadoop и MapReduce. Изучить основы Docker и Kubernetes для контейнеризации и оркестрации приложений. Ознакомиться с возможностями JCUDA для работы с GPU из Java.

Задание 1. Hadoop и MapReduce:

1. Установка Hadoop (в виртуальной машине или на сервере): Скачайте и установите Hadoop (например, в SingleNodeSetup). Настройте переменные окружения. Запустите Hadoop.

2. Загрузка данных в HDFS: Создайте директорию в HDFS. Загрузите текстовый файл (например, книгу в формате .txt) в HDFS.

3. Написание MapReduce программы (WordCount): Напишите Java-программу MapReduce, которая подсчитывает количество слов в текстовом файле, загруженном в HDFS. Скомпилируйте программу в JAR-файл.

4. Запуск MapReduce программы: Запустите программу MapReduce на кластере Hadoop, указав входные и выходные директории в HDFS.

5. Просмотр результатов: Просмотрите результаты работы MapReduce программы, хранящиеся в HDFS.

Задание 2. Docker и Kubernetes:

Скачайте и установите DockerDesktop (или DockerEngine на Linux). Создание Dockerfile для простого веб-приложения (например, на Python с Flask): Напишите простой веб-сервер на Python с использованием фреймворка Flask. Создайте Dockerfile, который определяет, как построить Docker-образ для этого приложения. В Dockerfile укажите базовый образ, установите зависимости и скопируйте код приложения. Сборка Docker-образа: Соберите Docker-образ на основе Dockerfile с помощью команды dockerbuild. Запуск Docker-контейнера: Запустите Docker-контейнер из созданного образа с помощью команды dockerrun . Проверьте, что веб-приложение доступно через браузер. Установка Minikube (для Kubernetes): Скачайте и установите Minikube. Развертывание приложения в Kubernetes: Создайте Deployment и Service для веб-приложения в Kubernetes. Используйте YAML-файлы для

описания Deployment и Service. Доступ к приложению в Kubernetes: Получите доступ к веб-приложению, развернутому в Kubernetes, через Minikube.

Задание 3.JCUDA (требует наличие GPU NVIDIA и установленных драйверов CUDA):

Скачайте и установите CUDA Toolkit от NVIDIA (если у вас есть совместимая видеокарта). Скачайте и установите JCuda библиотеки. Напишите Java-программу, которая умножает две матрицы, используя GPU через JCuda. Запустите программу и измерьте время выполнения умножения матриц на GPU. Реализуйте умножение матриц на CPU и сравните время выполнения с GPU.

Контрольные вопросы:

1. Что такое Hadoop и какие основные компоненты входят в его состав?
2. Объясните принцип работы MapReduce.
3. Что такое Docker и зачем он нужен?
4. Что такое Kubernetes и какие задачи он решает?
5. Что такое JCuda и как он позволяет использовать GPU из Java?
6. В чем преимущества использования контейнеризации с Docker и оркестрации с Kubernetes?
7. Какие задачи наиболее эффективно решать с использованием GPU и JCuda?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Код программ (MapReduce, Dockerfile, Java с JCuda).YAML-файлы для Kubernetes. Скриншоты, демонстрирующие результаты (запуск программ, вывод результатов, мониторинг Kubernetes). Результаты сравнения производительности (CPU vs GPU). Описание процесса настройки и проверки связи между устройствами. Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 7

Тема: **Ускорение параллельных вычислений на GPU.**

Цель: Изучить технологии CUDA и ROCm для программирования GPU. Освоить основные принципы параллельного программирования на GPU. Приобрести практические навыки разработки и оптимизации GPU-ускоренных приложений. Сравнить возможности и производительность CUDA и ROCm на различных задачах.

Задание 1. CUDA (для пользователей NVIDIA):

Напишите CUDA программу, которая выполняет сложение двух векторов на GPU.Определите ядро (kernel) для сложения элементов векторов. Выделите память на GPU для входных и выходных векторов. Скопируйте данные из хост-памяти (CPU) в память GPU.Запустите ядро на GPU.Скопируйте результаты из памяти GPU в хост-память. Освободите выделенную память на GPU.

Увеличьте размер блоков и гридов для достижения большей параллельности. Используйте общую память (sharedmemory) для уменьшения количества обращений к глобальной памяти. Измерьте время выполнения программы до и после оптимизации.

Используйте NVIDIA NsightSystems или NsightCompute для профилирования CUDA программы. Выявите узкие места в программе и предложите способы их устранения.

Задание 2. ROCm (для пользователей AMD):

Напишите HIP программу, которая выполняет сложение двух векторов на GPU. Используйте HIP API, которое позволяет коду работать как на CUDA, так и на ROCm. Определите ядро для сложения элементов векторов. Выделите память на GPU для входных и выходных векторов. Скопируйте данные из хост-памяти (CPU) в память GPU. Запустите ядро на GPU. Скопируйте результаты из памяти GPU в хост-память. Освободите выделенную память на GPU.

Увеличьте размер блоков и гридов для достижения большей параллельности. Используйте локальную память (localmemory) или кеш для улучшения производительности. Измерьте время выполнения программы до и после оптимизации.

Используйте ROCmprofiler для профилирования HIP программы. Выявите узкие места в программе и предложите способы их устранения.

Задание 3. Сравнение CUDA и ROCm (для пользователей с обеими видеокартами или возможностью использовать облачные сервисы):

Запустите программу сложения векторов на CUDA и ROCm с одинаковыми параметрами (размер векторов, количество блоков и потоков). Измерьте время выполнения программы на обеих платформах. Проанализируйте результаты и сделайте выводы о производительности CUDA и ROCm на данной задаче. Обсудите преимущества и недостатки CUDA и ROCm с точки зрения разработки и переносимости кода.

Контрольные вопросы:

1. Что такое GPU и чем он отличается от CPU?
2. Что такое CUDA и ROCm?
3. Объясните основные концепции CUDA (kernel, thread, block, grid, memory).
4. Как оптимизировать CUDA и ROCm программы?
5. Какие инструменты можно использовать для профилирования CUDA и ROCm программ?
6. Какие преимущества и недостатки CUDA и ROCm?
7. Что такое HIP и зачем он нужен?
8. Какие задачи наиболее эффективно решать с использованием GPU?

Отчет должен содержать:

Титульный лист (название работы, ФИО, группа). Цель работы. Краткое описание выполненных заданий. Код программ (CUDA и HIP).Результаты измерений времени выполнения (до и после оптимизации).Скриншоты с результатами профилирования. Анализ производительности и сравнение CUDA и ROCm (если возможно).Ответы на контрольные вопросы. Выводы о проделанной работе.

Лабораторная работа 8

Тема: **Подготовка и реализация проекта**.

Цель: Разработать и реализовать проект, демонстрирующий применение системы контроля версий (Git) для совместной разработки и методы распределенных вычислений для обработки больших объемов данных. Приобрести опыт планирования и организации работы над проектом в команде. Углубить знания в области Git, фреймворков для распределенных вычислений (Hadoop, Spark), контейнеризации (Docker) и оркестрации (Kubernetes). Научиться интегрировать различные технологии для решения комплексной задачи.

Общая концепция проекта:

Проект предполагает разработку системы, которая:

1. Собирает данные (например, логи веб-сервера, данные социальных сетей, результаты научных экспериментов).
2. Хранит данные в распределенной файловой системе (HDFS).
3. Обрабатывает данные с использованием фреймворка для распределенных вычислений (MapReduce, Spark).
4. Визуализирует результаты обработки.
5. Все этапы проекта управляются с помощью системы контроля версий (Git).

Этапы выполнения проекта:

1. Планирование проекта
2. Настройка окружения разработки
3. Разработка модулей проекта
4. Тестирование и отладка
5. Развертывание и масштабирование (опционально)
6. Документирование проекта

Пример проекта:

Система анализа логов веб-сервера

1. Сбор данных: Сбор логов веб-сервера (например, Apache или Nginx).
2. Хранение: Хранение логов в HDFS.

3. Обработка: Обработка логов с помощью Spark для анализа посещаемости сайта, выявления популярных страниц, обнаружения атак и т.д.
4. Визуализация: Визуализация результатов анализа с помощью графиков и диаграмм (например, с использованием библиотек Python: Matplotlib, Seaborn).

Задания к лабораторной работе:

1. Сформировать команду разработчиков (2-4 человека).
2. Разработать детальный план проекта (описание, архитектура, задачи, роли, сроки).
3. Создать репозиторий Git для проекта (на GitHub, GitLab или Bitbucket).
4. Реализовать все модули проекта (сбор данных, хранение, обработка, визуализация).
5. Использовать Git для контроля версий кода (ветвление, слияние, коммиты).
6. Написать тесты для проверки работоспособности модулей.
7. Задокументировать проект (техническая документация, руководство пользователя).
8. Подготовить презентацию проекта и продемонстрировать его работу.

Учебно-методическое дисциплины

Перечень основной литературы:

1. Гоетц, Б. Java Concurrency на практике / Б. Гоетц, Т. Пейер, Дж. Блох [и др.]; пер. с англ. А. Киселева. – СПб.: Питер, 2019. – 384 с. – ISBN 978-5-4461-1077-3.
2. Ченг Дж. Профессиональное программирование на CUDA C / Дж. Ченг, М. Гроссман, Т. Маккерчер. – М.: ДМК Пресс, 2015. – 480 с. – ISBN 978-5-97060-293-8.
3. Петрухнова Г. В. Введение в распределенные системы: учебное пособие / Г. В. Петрухнова. - Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2021. - 81 с. - ISBN 978-5-7731-0925-9. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. — URL: https://www.iprbookshop.ru/111462.html.

Перечень дополнительной литературы:

1. Таненбаум Э. Распределенные системы: принципы и парадигмы / Э. Таненбаум, М. ван Стеен. – 3-е изд. – СПб.: Питер, 2019. – 848 с. – ISBN 978-5-4461-1156-5.
2. Клейнберг Дж. Алгоритмы: разработка и применение / Дж. Клейнберг, Э. Тардос. – М.: Вильямс, 2016. – 880 с. – ISBN 978-5-8459-2016-0.
3. Уилсон Г. Проектирование распределенных систем: шаблоны и парадигмы / Г. Уилсон. – М.: Вильямс, 2021. – 480 с. – ISBN 978-5-8459-2100-6.
4. Шафран Дж. Kubernetes в действии / Дж. Шафран. – М.: ДМК Пресс, 2020. – 512 с. – ISBN 978-5-97060-800-8.
5. Уайт Т. Hadoop: подробное руководство / Т. Уайт. – 4-е изд. – М.: Вильямс, 2019. – 768 с. – ISBN 978-5-8459-2105-1.
6. Сандерс Дж. CUDA на примерах: введение в программирование GPU общего назначения / Дж. Сандерс, Э. Кандрот. – М.: ДМК Пресс, 2011. – 320 с. – ISBN 978-5-94074-660-7.
7. Кулурис Дж. Распределенные системы: концепции и проектирование / Дж. Кулурис, Ж. Доллимор, Т. Киндберг, Г. Блэр; пер. с англ. – 5-е изд. – М.: Вильямс, 2012. – 1120 с. – ISBN 978-5-8459-1754-2.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. https://git-scm.com - документация Git.
2. https://metanit.com/java/tutorial/ - туториал по языку программирования Java.
3. https://docs.nvidia.com/cuda/cuda-c-programming-guide/ - документация по технологии Cuda.
4. http://www.jcuda.org/jcuda/JCuda.html - документация по библиотеке JCuda.

5. https://blogs.oracle.com/javamagazine/post/programming-the-gpu-in-java - небольшой обзор JCuda
6. https://rocm.docs.amd.com/en/latest/ - документация по технологии ROCm.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по организации и проведению самостоятельной работы
по дисциплине
«РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ»
для студентов направления подготовки
02.03.01 Математика и компьютерные науки
направленность (профиль)
«Анализ данных и искусственный интеллект»

Общие положения

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

– формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);

– написание докладов;

– подготовка к семинарам, практическим и лабораторным работам, их оформление;

– составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

– выполнение учебно-исследовательских работ, проектная деятельность;

– подготовка практических разработок и рекомендаций по решению проблемной ситуации;

– выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;

– компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;

– выполнение курсовых работ (проектов) в рамках дисциплин;

– выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине **«Распределенные вычисления»** направлена на формирование следующих **компетенций**:

- **ПК-1.** Способен демонстрировать базовые знания математических и естественных наук, основ программирования и информационных технологий
- **ПК-7.** Способен подготавливать данные и проводить аналитические работы по исследованию больших данных

1. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование соответствующего набора компетенций будущего бакалавра по направлению подготовки «Математика и компьютерные науки».

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

2. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
6 семестр					
ПК-1, ПК-7	Самостоятельное изучение литературы	Собеседование	18	2	20
ПК-1, ПК-7	Подготовка к лабораторным работам	Отчет	38	2	40
Итого за 6 семестр			**56**	**4**	**60**
Итого			**56**	**4**	**60**

3. Порядок выполнения самостоятельной работы студентом

3.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют ***четыре основные установки в чтении научного текста***:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

3.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выво-

дом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

3.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

3.4. Методические рекомендации по написанию научных текстов (докладов, рефератов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т, п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Реферат (доклад) - это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Реферат не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в реферате должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки реферата студентом.

Выполнение реферата начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания реферата. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста реферата предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки реферат сдается на кафедру для его оценивания руководителем.

Требования к написанию реферата

Написание 1 реферата является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема реферата может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Реферат должен быть написан научным языком.

Объем реферата должен составлять 20-25 стр.

Структура реферата:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.
- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.
- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса
- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.

- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;
- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- реферат должен быть представлен в сброшюрованном виде.

Порядок защиты реферата:

Защита реферата проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту реферата отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите реферата приветствуется использование мультимедиа-презентации.

Оценка реферата

Реферат оценивается по следующим критериям:

• соблюдение требований к его оформлению;

• необходимость и достаточность для раскрытия темы приведенной в тексте реферата информации;

• умение студента свободно излагать основные идеи, отраженные в реферате;

• способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

Темы рефератов по дисциплине «Распределенные вычисления»

1. Особенности современных распределённых вычислительных систем.
2. Оптимизация структуры распределённой вычислительной системы.
3. Моделирование распределённой вычислительной системы.
4. Синтез алгоритма структурной оптимизации распределённой вычислительной системы.
5. Анализ влияния параметров разработанного алгоритма декомпозиции на эффективность проектирования распределённой вычислительной системы.
6. Архитектура федеральных и территориальных региональных распределённых вычислительных систем.
7. Построение распределённой вычислительной системы на основе сети рабочих станций средствами программного инструментария Condor.
8. Построение распределённой вычислительной системы на основе сети рабочих станций средствами программного инструментария Legion.
9. Разработка инструментария для построения распределённой вычислительной системы на основе персональных компьютеров, подключённых к сети Интернет.
10. Разработка программ численного моделирования для работы в среде NumGRID.
11. Анализ производительности коммуникаций между вычислительными узлами в среде NumGRID и формулирование рекомендаций к оптимизации прикладных программ для NumGRID и системных средств NumGRID

Список литературы для СРС

Перечень основной литературы:

1. Гоетц, Б. Java Concurrency на практике / Б. Гоетц, Т. Пейер, Дж. Блох [и др.]; пер. с англ. А. Киселева. – СПб.: Питер, 2019. – 384 с. – ISBN 978-5-4461-1077-3.
2. Ченг Дж. Профессиональное программирование на CUDA C / Дж. Ченг, М. Гроссман, Т. Маккерчер. – М.: ДМК Пресс, 2015. – 480 с. – ISBN 978-5-97060-293-8.
3. Петрухнова Г. В. Введение в распределенные системы: учебное пособие / Г. В. Петрухнова. - Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2021. - 81 с. - ISBN 978-5-7731-0925-9. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. — URL: https://www.iprbookshop.ru/111462.html.

Перечень дополнительной литературы:

1. Таненбаум Э. Распределенные системы: принципы и парадигмы / Э. Таненбаум, М. ван Стеен. – 3-е изд. – СПб.: Питер, 2019. – 848 с. – ISBN 978-5-4461-1156-5.
2. Клейнберг Дж. Алгоритмы: разработка и применение / Дж. Клейнберг, Э. Тардос. – М.: Вильямс, 2016. – 880 с. – ISBN 978-5-8459-2016-0.
3. Уилсон Г. Проектирование распределенных систем: шаблоны и парадигмы / Г. Уилсон. – М.: Вильямс, 2021. – 480 с. – ISBN 978-5-8459-2100-6.
4. Шафран Дж. Kubernetes в действии / Дж. Шафран. – М.: ДМК Пресс, 2020. – 512 с. – ISBN 978-5-97060-800-8.
5. Уайт Т. Hadoop: подробное руководство / Т. Уайт. – 4-е изд. – М.: Вильямс, 2019. – 768 с. – ISBN 978-5-8459-2105-1.
6. Сандерс Дж. CUDA на примерах: введение в программирование GPU общего назначения / Дж. Сандерс, Э. Кандрот. – М.: ДМК Пресс, 2011. – 320 с. – ISBN 978-5-94074-660-7.

7. Кулурис Дж. Распределенные системы: концепции и проектирование / Дж. Кулурис, Ж. Доллимор, Т. Киндберг, Г. Блэр; пер. с англ. – 5-е изд. – М.: Вильямс, 2012. – 1120 с. – ISBN 978-5-8459-1754-2.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. https://git-scm.com - документация Git.
2. https://metanit.com/java/tutorial/ - туториал по языку программирования Java.
3. https://docs.nvidia.com/cuda/cuda-c-programming-guide/ - документация по технологии Cuda.
4. http://www.jcuda.org/jcuda/JCuda.html - документация по библиотеке JCuda.
5. https://blogs.oracle.com/javamagazine/post/programming-the-gpu-in-java - небольшой обзор JCuda
6. https://rocm.docs.amd.com/en/latest/ - документация по технологии ROCm.